Appeared in the International Symposium on Experimental Robotics (ISER'99),
Sydney, Australia, March 25-28,1999.

RoverBug: Long Range Navigation for Mars Rovers

Sharon Laubach and Joel Burdick
California Institute of Technology
Pasadena, CA, USA
Sharon.Laubach@jpl.nasa.gov / jwb@robby.caltech.edu

Abstract: After Mars Pathfinder's success, a demand for new mobile robots for planetary exploration arose. These robots must be able to autonomously traverse long distances over rough, unknown terrain, under severe resource constraints. We present the "RoverBug" algorithm, which is complete, correct, requires minimal memory, and uses only on-board sensors, which are guided by the algorithm to sense only the data needed for navigation. The implementation of RoverBug on the Rocky7 Mars Rover prototype at the Jet Propulsion Laboratory (JPL) is described, and experimental results from operation in simulated martian terrain are presented.

1. Introduction

The Mars Pathfinder mission illustrated the benefits of including a mobile robotic explorer on a planetary mission. While previous forays allowed only remote exploration or were limited to a small site by an immobile lander, the Sojourner rover was able to roam and to place its instruments on objects of interest. Mars missions currently being planned call for new rovers capable of operation for up to a year, compared to the 83 sols (martian days) of operation for the Sojourner rover. The rovers are also required to traverse vastly greater distances: up to 100m/sol, versus Sojourner's 104m/83 sols. Lessons learned from Mars Pathfinder indicate a need for significantly increased rover autonomy in order to meet mission criteria within severe constraints including limited communication opportunities with Earth, power, and computational capacity.

1.1. Motion Planning on Mars

A key advance in functionality required for planetary rovers is greater navigational autonomy. Given a goal which cannot be seen from the rover's location, the rover must use its sensors to navigate safely and accurately through unknown, rough terrain to that goal, autonomously. This will require, in particular, improved motion planning and localisation algorithms.

Navigation techniques for planetary rovers must assume no prior knowledge of the environment and must be sensor-based and robust. They must also operate under severe constraints of power, computational resources, and memory, due to the high cost of flight components. Due to dead reckoning errors, slippage, nonholonomic fine-positioning constraints, and constraints on mission time available, using rover motion to augment sensing is costly. Simultaneously, limited memory, computational capacity, power and time available all argue for minimising the amount of data sensed and processed. Thus, practical navigation techniques must utilise the available sensing array in a scheme which efficiently senses only the data needed for navigation, requires minimal memory to store salient features of the environs, and conserves rover motion.

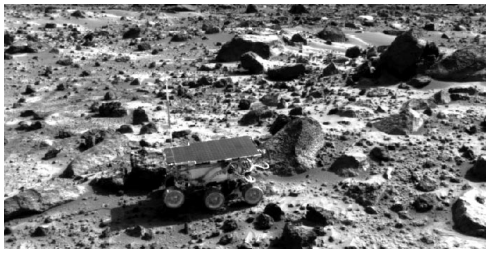

Figure 1. Typical terrain encountered on Mars by the Sojourner rover.

2. Relevant Work

Much of the work in motion planning can be divided into three major categories: "classical" path planners, heuristic planners, and "complete and correct" sensor-based motion planners. "Classical" planners assume full knowledge of the environment, and are complete. Heuristic planners, generally based upon a set of "behaviours," can be used in unknown environments but may generate long paths and do not guarantee the goal will ever be reached, nor that the algorithm will halt. (A more detailed discussion is given in [1].) The third category, which relies solely upon the rover's sensors and yet guarantees completeness, is most relevant to the problem of autonomous planetary navigation.

Two approaches to such planners adapt classical methods to local sensed areas. One technique builds "roadmaps" using data from the visible region, such as Choset's HGVG [2], Rimon's adaptation of Canny's OPP [3], and the TangentBug algorithm of Kamon, Rivlin, and Rimon [4]. The second approach springs from approximate cell decomposition, filling in a grid-based world model as information is gathered, exemplified by Stentz' D* algorithm [5]. These methods differ in their level of development for real systems. The sensor-based version of OPP is currently strictly theoretical, owing to the difficult-to-implement sensors required. The HGVG has been implemented on a mobile robot using sonar sensors. This planner produces paths which are maximally distant from obstacles, a plus for rover safety, but it works best in a contained area—a description not applicable to the typical martian environment (Fig. 1).

The D* algorithm and TangentBug are both useful in unbounded environments, and both produce "locally optimal" solutions: the resultant paths are the shortest length possible using only local information. D* has been implemented on an autonomous HMMWV; however, its grid-based world model requires a significant amount of memory for storage, and the algorithm's completeness depends upon the cell granularity of its world model.

TangentBug motivates the work presented here. Its world model is streamlined, comprising primarily the sensed obstacle endpoints. The planner consists of two "modes", motion-to-goal and boundary following, which interact to ensure global convergence if the goal is reachable, and which "fail gracefully" if the goal is unreachable. The algorithm is memory-efficient, fairly robust, and conserves robot motion. However, some of its assumptions do not apply to the "rover problem" of navigating in planetary terrain: TangentBug assumes that the robot is a point, that obstacles block both motion and sensing, and that the robot's sensors provide an omnidirectional view.

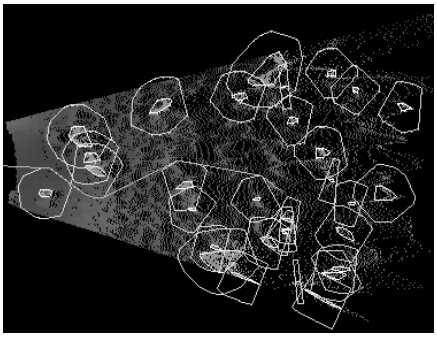

Figure 2. Rangemap data from a stereo pair. This image also shows detected obstacles, and a path generated by the RoverBug algorithm (see Section 4).

The current plan for a rover sensing system consists of a mast-mounted stereo pair of cameras that can pan and tilt. These cameras have a 30° to 45° field of view (FOV), and the "visible region" associated with these sensors sweeps out roughly a wedge, with limited downrange radius R due to both the tilt angle and camera resolution. (Fig. 2 shows data from such a sensing array.) The rovers also feature chassis-mounted stereo pairs on the front and back. Given the constraints described above, we cannot simply pan the mast-mounted sensor array and combine many views to obtain a 360° view at each step. Rather, the planner should be able to identify the minimal number of sensor readings needed (and which specific areas to sense) to proceed at each step, while avoiding unnecessary rover motion. Thus, we have developed the "Wedgebug" algorithm and its extension, "RoverBug", to address these issues for flight microrovers. Wedgebug deals with the limited FOV of rovers in an efficient manner, minimising the need to sense and store data, using autonomous gaze control. The RoverBug implementation discussed in Section 4 relaxes the assumptions that the rover is a point robot, and that obstacles block sensing.

3. The Wedgebug Algorithm

Wedgebug assumes the following: The rover is modelled as a point robot in a 2D world where every point is either contained within an impassable obstacle or lies in freespace ($\mathfrak{F}$). Obstacles block both motion and sensing. The rover's sensors, from position x, detect ranges within a wedge $W_i = W(x, \vec{v}_i)$ which sweeps out an angle 2α and is centered on the direction $\vec{v}_i$, where $\angle(\overrightarrow{xT}, \vec{v}_i) = 2i\alpha$ (T is the goal). Define C as the arc boundary of W_i at radius R, and ∂W_i as the union of the two bounding rays of W_i (Fig. 3). The "interior" of W_i is defined as $\text{int}(W_i) = \overline{W_i} - \partial W_i$ (an "interior" point may lie on C). Let $\text{d}(a, b)$ be the Euclidean distance between points a and b.

Wedgebug, like TangentBug, has two modes which interact to ensure global convergence: *motion-to-goal* (*MtG*) and *boundary following* (*BF*). Each mode is further divided into components to improve efficiency and handle the limited FOV. At the robot's initial position A, an initialisation step records the parameter $\text{d}_{\text{LEAVE}} = \text{d}(A, T)$. This parameter marks the farthest the robot can stray from T during an *MtG* segment. Thereafter, *MtG* is typically the dominant

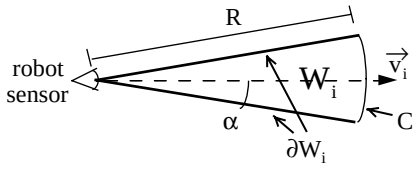

Figure 3. Anatomy of a wedge.

behaviour. It directs the robot to move toward the goal using a local version of the tangent graph, restricted to the visible area (Fig. 4). The robot (at position x) first senses a wedge, W_0, directed toward the goal. The tangent graph (or "reduced visibility graph") is constructed, consisting of all line segments in $\mathfrak{F}$ connecting x, T, and all obstacle vertices such that the segments are tangent to any obstacles they encounter [6]. The *local tangent graph* within the wedge W, LTG(W), is defined as the tangent graph restricted to W. (Obstacle boundaries appear as continuous contours in the range data; the endpoints of these contours are called the "obstacle vertices". Each endpoint e corresponds to a discontinuity in the range data or to an intersection of a contour with ∂W or C.) The planner constructs LTG(W_0). The planner optionally adds a node, the projection of T onto C, so LTG(W_0) contains a path directly towards T. The planner then searches a subgraph, $G1(W_0)$, consisting of those nodes closer to T than both the robot's current position and $\mathrm{d}_{\mathrm{LEAVE}}$, for the optimal local subpath. Using the criteria in Section 3.1, the robot may scan additional wedges, construct the LTG in the conglomerate wedge $\overline{W}(x)$ (see Section 3.1), and search for a new subpath. After executing the resultant subpath, *MtG* begins anew. This behaviour is continued until either the goal is reached, T is deemed unreachable, or the robot encounters a local minimum in $\mathrm{d}(\cdot, T)$. In the latter case, the planner switches to *BF*. The objective of this mode is to skirt the boundary of the obstacle which contains the local minimum, still calculating LTG(W_0), until one of two events occur: either the robot completes a loop, in which case T is unreachable and the algorithm halts; or LTG(W_0) contains a new subpath toward T and the planner switches back to *MtG*. Based upon the two operational modes, *MtG* and *BF*, it can be proved that Wedgebug is complete and correct [7]. Practically, by implementing a form of gaze control, the algorithm also deals with the limited FOV of flight rovers in a manner which is efficient and minimises the need to sense and store data. Furthermore, Wedgebug produces locally optimal paths. Hence, it is well suited to conserving rover energy. The next two sections describe the *MtG* and *BF* modes in more detail. (See [7] for a thorough description.)

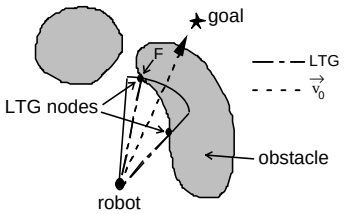

Figure 4. LTG calculated within $W(x, \vec{v}_0)$.

3.1. Motion-to-Goal

The goal of *MtG* is to move so the robot's distance to T is nonincreasing. During *MtG*, the robot can either move through $\mathfrak{F}$ toward the goal ("direct mode"), or it must skirt the boundary of an obstacle while moving toward T ("sliding mode"). Further, "sliding mode" contains a submode, "virtual *MtG*," to improve efficiency. That is, during normal *MtG*, the planner scans a single wedge toward T to determine whether a path exists. If it is apparent that more information could lead to a shortcut, "virtual *MtG*" scans additional wedges in order to determine the appropriate path.

The first actions taken in a new *MtG* step are to scan W_0, construct LTG(W_0), and search $G1(W_0)$. The shortest local path P will either aim directly toward T, or will pass through an endpoint e of a *blocking obstacle* (which lies directly between x and T). Call the point through which P passes—either e, or the projection of T on C (T_g)—the *focus point*, F (Fig. 4). The focus point (fixed for each step) is the goal for each *MtG* step: the subpath for the current *MtG* step is precisely $\overline{xF}$. Its position within the robot's FOV determines whether additional wedge views are needed. Basically, if the subpath moves the robot through the interior of the visible region, the robot executes this subpath and begins the next *MtG* step. If, on the other hand, F lies on the obstacle boundary, an additional view in this direction could produce a shortcut around the blocking obstacle—that is, the robot could "virtually circumnavigate" a portion of the boundary without moving (see Fig. 5). The "virtual *MtG*" mode ends when 1) the robot has found a suitable shortcut, so the robot moves along this subpath and begins the next *MtG* step; 2) the rover detects that it is sensing part of a region not useful for *MtG*, i.e. farther than the rover from T; or 3) the robot detects that the obstacle boundary is curving away from T, so the robot can no longer "virtually slide" in this direction without losing ground. In the latter cases, if the rover has not yet established a traversal direction, the robot attempts to round the obstacle in the opposing direction. If this attempt fails, the robot has encountered a local minimum in $d(\cdot, T)$, and the planner switches to *BF*.

In order to prevent backtracking while circumnavigating an obstacle O (both "virtually" and while moving), the algorithm establishes a traversal direction—call it *positive* (ρ^+)—upon first sensing O. Thereafter, the robot must round the obstacle in only the positive direction, until (1) the blocking obstacle is changed (including the case when $F = T_g$), (2) the robot detects

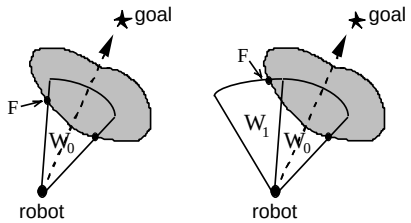

Figure 5. "Virtual MtG." The figure on the left shows the first part of an MtG step. The nodes of LTG(W_0) are marked. F satisfies the conditions for "virtual MtG," so the robot scans W_1 (right). Now, $F \in int(W_0 \cup W_1)$, so "virtual MtG" ends.

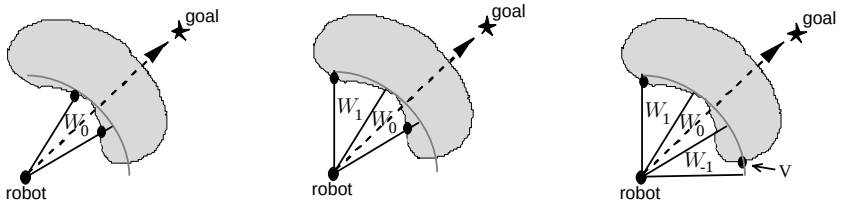

Figure 6. "Virtual BF." The figure on the left depicts the first part of a "virtual BF" step. The nodes of LTG(W_0) are marked. Since $\nexists V \in int(\mathrm{LTG}(W_0))$, the robot scans W_1 (center). Again, $\nexists V \in int(\mathrm{LTG}(W_0 \cup W_1))$, so the robot scans W_{-1} (right). Now, $V \in int(W_0 \cup W_1 \cup W_{-1})$, so "virtual BF" ends.

that it has completely circumnavigated O and the algorithm halts , or (3) the planner switches to *BF*. The latter case occurs when the robot can no longer decrease its distance to T—i.e, it has entered the basin of attraction of a local minimum in $\mathrm{d}(\cdot, T)$.

3.2. Boundary Following

The goal of *BF* is to skirt the blocking obstacle O until progress can be made once more toward T, thereby escaping a local minimum. As with *MtG*, *BF* is split into two submodes. Normal *BF* uses two wedge views, one toward T and one in the direction of travel around the obstacle boundary (ρ^+), to determine when a path towards T appears while the robot circumnavigates O. Immediately after a switch from *MtG* to *BF*, however, ρ^+ may not be defined. In this case, "virtual *BF*" is used to take advantage of the information from the current distance from O to choose this direction wisely. (The motivation for "virtual *BF*," as for "virtual *MtG*," is the idea that it is less costly for the robot to swivel its sensors than for the robot to move.) In essence, the robot swings its sensors back and forth in a prescribed manner to search for the "best" place to move and begin normal *BF*.

More precisely, the robot initially scans $W_1 = W(x, \vec{v}_1)$, where the positive direction is chosen by comparing the tangents to ∂O at the wedge boundary. The "steepest" side is chosen with the idea that the rover may be able to progress further "virtually" in this direction. As before, let $\overline{W} = \bigcup^{sensed} W_k(x)$. The planner computes $\mathrm{LTG}(\overline{W})$. Similarly to *MtG*, if a shortcut is found through the interior of $\overline{W}$, the robot moves along this path and begins normal *BF*, first recording d_{reach}, the closest distance to T found along ∂O, ρ^+, and V_{loop}, a marker used to detect whether the robot has looped around O. Otherwise, the planner directs the sensor to scan $W_{-1} = W(x, \vec{v}_{-1})$, constructs $\overline{W} = W_0 \cup W_1 \cup W_{-1}$, and searches the freshly expanded $\mathrm{LTG}(\overline{W})$. In this manner, the robot scans back and forth until a suitable shortcut is found, then travels it to begin normal *BF*. "Virtual *BF*" ends when one of three events are detected: 1) a suitable shortcut is found, so the robot moves along this subpath and begins normal *BF*; 2) the latest wedge overlaps a previously scanned region—the robot is trapped by an encircling obstacle, and the algorithm halts; and finally 3) "virtual *BF*" is no longer useful, since a second obstacle obscures the blocking obstacle; the point where the two boundaries "meet" is called the "framing point" V_f. If one such point is visible, the robot

scans once more in the opposing direction and "virtual *BF*" ends. If a shortcut as in (1) is found, the robot moves there to begin normal *BF*. Otherwise, the rover moves to the point on ∂O adjacent to V_f. If there are two such points, V_l and V_r, the rover moves to the point which is "farthest around" the obstacle, then begins normal *BF*.

To start a normal *BF* step, the robot senses W_0 and searches $G1(W_0)$. *BF* exits if: (1) $T \in W_0$, in which case the robot moves to T and the algorithm halts, or (2) $\exists V \in G1(W_0)$ such that $\mathrm{d}(V,T) < \mathrm{d}_{reach}$, the *leaving condition*, in which case the planner sets $\mathrm{d}_{\mathrm{LEAVE}}$ to $\mathrm{d}(V,T)$ and begins a new *MtG* segment. If neither of these conditions hold, the planner directs the sensor to scan a wedge W_x along the tangent to the obstacle boundary. If $V_{loop} \in W_x$ and it is in the sensed portion of ∂O containing x, the robot has executed a loop—the goal is unreachable, and the algorithm halts. Otherwise, the planner computes the farthest the robot can traverse along ∂O in W_x. The robot records d_{reach}, moves, and begins a new *BF* step.

3.3. Sketch of Proof of Convergence

The proof of convergence for Wedgebug is similar to TangentBug's proof [4]. Each robot motion can be characterised as a particular type of motion segment; in turn, each type of segment can be shown to have finite length. Following [4], it can be shown that there are a finite number of each type of segment, so the path terminates after finite length. Due to space limitations, we detail only the proof that *BF* segments have finite length. The proofs for the other types of motion segments are analagous.

Define S_i to be the point where the planner switches from *MtG* to *BF* at obstacle O_i; L_i the point where the *leaving condition* is met on O_i; and finally Q_i, the point where a loop is detected on O_i. There are two types of *BF* segments: $[S_i, L_i]$, and $[S_i, Q_i]$. Let P_i be the perimeter of obstacle O_i.

Lemma. *BF* segments are finite length.

Proof. a) $[S_i, L_i]$. Let N be the point where the robot first touches ∂O_i. The path then consists of two pieces: $[S_i, N]$ and $[N, L_i]$. Since N and L_i both lie on ∂O_i, the robot is traversing O_i in a fixed direction, and the robot has not detected a loop, we have $\mathrm{length}([N, L_i]) \leq \mathrm{length}([N, Q_i]) \leq \mathrm{length}([N, V_{loop}])$. Further, since $\underline{V_{loop}}$ is in the opposite direction from traversal, we know that the segments $\overline{NV_{loop}}$ and $\overline{V_{loop}N}$ do not overlap. Therefore, since P_i is finite, we have $\mathrm{length}([N, L_i]) \leq \mathrm{length}([N, V_{loop}]) \leq P_i < \infty$. Further, $\mathrm{d}(S_i, N) \leq R$. Thus, $\mathrm{length}([S_i, L_i]) \leq \mathrm{d}(S_i, N) + \mathrm{length}([N, L_i]) \leq R + P_i < \infty$.

b) $[S_i, Q_i]$. Similarly, $\mathrm{length}([S_i, Q_i]) \leq \mathrm{d}(S_i, N) + P_i \leq R + P_i < \infty$. □

4. Implementation and Results

An extended version of the Wedgebug algorithm, called "RoverBug," has been implemented on the JPL Rocky7 prototype microrover (Fig. 7), a research vehicle designed to test technologies for future missions [8]. Rocky7, which at 60cm x 50cm x 35cm is roughly the same size as the Sojourner rover, has a few important differences relevant to future rovers, including carrying three stereo

pairs of cameras for navigation: two body mounted, and one on a deployable 1.2m mast. In addition, the rover software features a localisation algorithm utilising mast imagery to aid in dead-reckoning [9].

Although the Wedgebug algorithm is an important step, it still does not capture the complexities of the real world. For instance, the rover is not a point robot; a problem addressed in the RoverBug implementation by calculating the obstacles' "silhouettes": the smallest polygon bounding the projection of each $SE(2)$ obstacle onto $\Re^2$. Another issue arises since the mast imagery can "see over" many obstacles: the resulting visible region is not a star-shaped set, and the LTG is much richer than in the development in Section 3. Also, the mast is limited in its ability to sense obstacles within 1m of the rover, since the obstacle detection algorithm searches for steps in elevation, not easy to detect while looking straight down on the tops of rocks. Thus, care must be taken while executing the subpaths.

The experimental scenario is as follows: Rocky7 is situated in unknown, rough terrain. The remote human operator views panoramic imagery returned by the rover, or orbiter and/or descent imagery to designate a goal coordinate. The operator then transmits the command to navigate to the goal, which sets in motion the autonomous planner. RoverBug begins by directing the mast to image towards the goal. Software on-board produces a rangemap, detects obstacles, and computes the obstacles' convex hulls. RoverBug then computes the obstacles' silhouettes, and searches the resulting LTG—which now truly looks the part of a local tangent graph—to produce the first subpath (see Fig. 2). The planner directs the mast to look in the appropriate direction(s), and incrementally builds and executes each subpath until the goal is reached.

As before, the motion-to-goal (MtG) mode is the dominant behaviour, moving the rover toward T monotonically, and boundary following (BF) is used to escape local minima in $\mathrm{d}(\cdot, T)$. However, rather than endowing each of these modes with a "virtual" submode, we combine the two submodes into a single "virtual sliding" (VS) mode in the interest of reducing the implementation's footprint, helping further minimise memory requirements. Upon detecting an obstacle chain which prevents progress through the single wedge view used by MtG to move toward the goal, the robot invokes VS mode. The purpose of

Figure 7. The Rocky7 Prototype Microrover. It is pictured in the JPL MarsYard, an outdoor testing arena featuring simulated martian terrain.

VS mode is to scan one wedge at a time, back and forth, until enough of the (visible) extent of the blocking obstacle has been seen[1] in order to determine which direction the robot should circumnavigate the obstacle chain. (Such a chain can be seen spanning the end of the wedge in Fig. 2.) At this point, RoverBug plans a path through the cumulative LTG (at x) to the appropriate edge of the blocking obstacle (chain) boundary, and enters *BF* mode.

BF mode, in turn, relies upon the body-mounted stereo cameras to gather information about obstacles, for two reasons: 1) the rover is likely too close for mast imagery to be useful, and 2) since the mast should not be raised while the rover moves, it would adversely affect the time required to navigate if the mast were raised and lowered between each (small) *BF* step. Since these cameras are fixed to the rover chassis, it is necessary to rotate the rover itself to ensure views in the necessary directions. As with Wedgebug, each *BF* step begins with a view toward the goal. If no obstacles are detected in this direction, then the rover raises its mast to determine whether the *leaving condition* (same as for Wedgebug) is satisfied. If the leaving condition holds, the rover switches back to *MtG*. Otherwise, the rover turns in the direction of circumnavigation, images the new region (with the body-mounted cameras), and determines whether another view is required to advance as far as possible around the obstacle boundary. In this wise, the rover uses its body cameras to skirt the obstacle chain, taking shortcuts where possible, until either the rover can again progress toward the goal, or until the robot has detected a loop. As before, a loop indicates that the goal is unreachable, and the algorithm halts.

Finally, in order to cope with the limitations of dead reckoning, the localisation algorithm described in [9] brackets each *MtG* step, to update the rover's knowledge of its position after executing each subpath, and each *BF* segment, to relocalise the rover after it has skirted an obstacle boundary. (If the boundary is particularly long, the rover may execute several localisation events along the way, due to the short accuracy range of the localisation algorithm.)

The *MtG* mode of RoverBug has been tested extensively in the JPL MarsYard, as well as in natural arroyo terrain, including traverses for tens of meters requiring multiple iterations of the motion planning and localisation algorithms. Figure 8 shows the processed experimental data from a 21m traverse in the MarsYard. Each "wedge" in the figure depicts a projected view of the rangemap computed from a stereo pair of (mast) images taken by the rover during its traverse. That is, each wedge represents the rover's FOV, containing terrain height information. The polygons represent the inaccessible regions around rocks deemed too large for rover traversal. The path superimposed on the figure is the path taken by the rover to avoid the rock obstacles and reach the goal. The "jags" in the path represent localisation events where dead reckoning is updated using the algorithm described in [9]. As in Fig. 2, the silhouettes are computed within each wedge view, and a subpath generated, which is executed before the next wedge view is taken. The resultant multi-step path runs from left to right.

[1] Note that in the RoverBug scenario, the frequency of "framing nodes" is significantly reduced, since the rover can "see over" most obstacles.

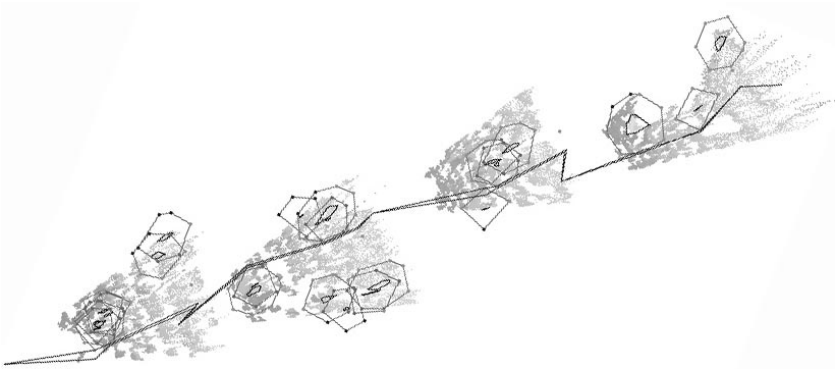

Figure 8. Results from the JPL MarsYard. The path runs left to right. Each wedge is a rangemap from mast imagery, extending roughly 5m from the imaging position.

5. Conclusion

The requirements for autonomous flight rovers for planetary exploration provide compelling motivation for work in sensor-based navigation. This paper continues the work begun in [1] to develop, implement, and test a robust, practical path planner for the Rocky7 prototype microrover. We believe that the RoverBug planner will significantly augment microrovers' autonomous navigation ability, which in turn will aid in producing successful mobile robot missions.

Acknowledgments

The work described here was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. The authors acknowledge the Long Range Science Rover team, particularly Samad Hayati, and the Mars Pathfinder Rover team, for help, inspiration, and flight experience with a rover.

References

[1] Laubach S L, Burdick J W, Matthies L H 1998 An autonomous path-planner implemented on the Rocky7 prototype microrover. *Proc IEEE Conf Rob. Automat.*

[2] Choset H 1996 Sensor based motion planning: the Hierarchical Generalized Voronoi Graph. Ph.D. thesis, California Inst of Tech

[3] Rimon E, Canny J 1994 Construction of c-space roadmaps from local sensory data: what should the sensors look for? *Proc IEEE Conf Rob. Automat.*

[4] Kamon I, Rimon E, Rivlin E 1995 A new range-sensor based globally convergent navigation algorithm for mobile robots. CIS–Center of Intelligent Systems 9517, Computer Science Dept, Technion, Israel

[5] Stentz A 1994 Optimal and efficient path planning for partially-known environments. *Proc IEEE Conf Rob. Automat.*

[6] Latombe J-C 1991 *Robot Motion Planning.* Kluwer Academic Publishers, Boston

[7] Laubach S L 1999 A practical autonomous sensor-based path planner for flight planetary microrovers. Ph.D. thesis, California Inst of Tech

[8] Volpe R, Balaram J, Ohm T, Ivlev R 1996 The Rocky7 Mars Rover prototype. *Proc IEEE/RSJ Conf Intelligent Robots and Sys.*

[9] Olson C, Matthies L 1998 Maximum likelihood rover localisation by matching range maps. *Proc IEEE Conf Rob. Automat.*